Neural Networks
HW1 - Convolutional Neural Networks (CNNs)

Cătălin-Alexandru Rîpanu IA1-A
Automatic Control and Computers
National University of Science and Technology POLITEHNICA Bucharest
E-mail:catalin.ripanu@upb.ro
Lecturer: Alexandru Sorici

12th November, 2024

Abstract

This report presents a comprehensive analysis and implementation of convolutional neural networks (CNNs) for image classification tasks. The work encompasses three main components: manual implementation of CNN forward pass operations, development of a custom CNN architecture with various regularization techniques, and transfer learning using a pre-trained ResNet-18 model. The study evaluates different approaches to improve model performance, including batch normalization, dropout, and data augmentation, achieving the target of over 60% test accuracy on the Imagenette dataset.

1 Introduction

Convolutional Neural Networks have revolutionized computer vision tasks through their ability to automatically learn hierarchical feature representations from image data. This assignment explores three fundamental aspects of CNNs:

1. Implementation of basic CNN operations from scratch
2. Design and optimization of a custom CNN architecture
3. Application of transfer learning techniques using pre-trained models

The primary objective is to achieve superior classification performance on the Imagenette dataset while gaining practical understanding of CNN principles and optimization strategies.

2 Task 1: Manual Implementation of CNN Operations

2.1 Convolution Forward Pass

The first task involved implementing the forward pass of a convolutional layer from scratch. This implementation was done in the `MyConvStub` class:

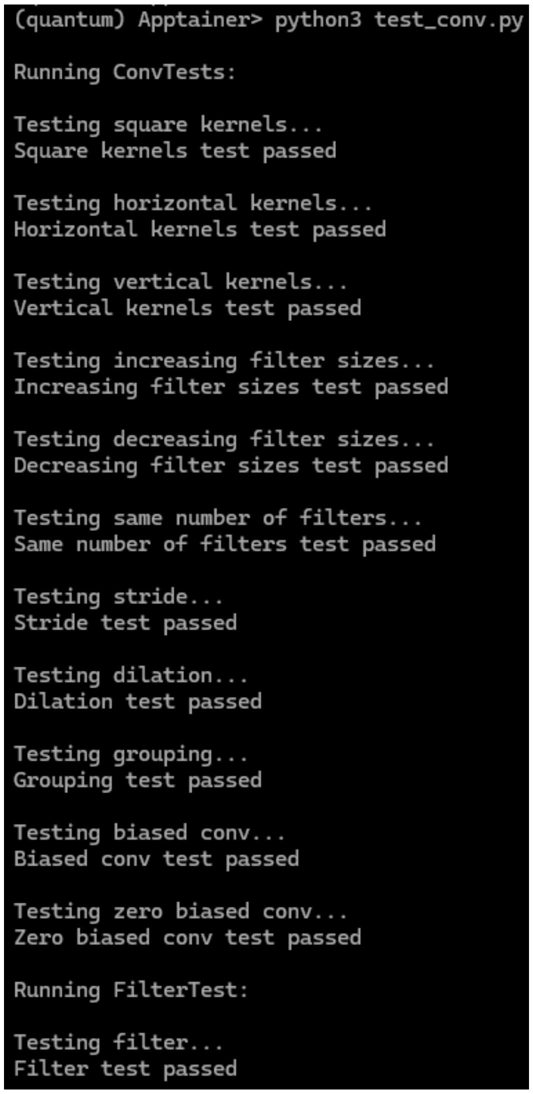
- Configurable kernel size and stride
- Support for dilated convolutions
- Group convolution functionality
- Multi-channel input handling

2.2 Filter Operations

A separate `MyFilterStub` class was implemented to handle 2D filter operations across multiple input channels. These are the features implemented:

- Channel-wise filter application
- Proper handling of padding
- Support for various filter types

2.2.1 Tests Validation



```
(quantum) Apptainer> python3 test_conv.py

Running ConvTests:

Testing square kernels...
Square kernels test passed

Testing horizontal kernels...
Horizontal kernels test passed

Testing vertical kernels...
Vertical kernels test passed

Testing increasing filter sizes...
Increasing filter sizes test passed

Testing decreasing filter sizes...
Decreasing filter sizes test passed

Testing same number of filters...
Same number of filters test passed

Testing stride...
Stride test passed

Testing dilation...
Dilation test passed

Testing grouping...
Grouping test passed

Testing biased conv...
Biased conv test passed

Testing zero biased conv...
Zero biased conv test passed

Running FilterTest:

Testing filter...
Filter test passed
```

Figure 1: Tests output

3 Task 2: Custom CNN Architecture

3.1 Dataset Analysis

The Imagenette dataset consists of 10 classes with images resized to 160x160 pixels. The dataset includes classes such as:

- Tench
- English Springer
- Cassette Player
- Chain Saw
- Church
- French Horn
- Garbage Truck
- Gas Pump
- Golf Ball
- Parachute

3.2 Model Architecture

The custom CNN architecture consists of:

- Three convolutional layers with ReLU activation
- Max pooling layers for dimensionality reduction
- Two fully connected layers
- Batch normalization layers (in specific configurations)
- Dropout layers (in specific configurations)

3.3 Training Configurations

Five distinct model configurations were implemented and evaluated to understand the impact of different regularization techniques:

1. **Baseline Model**
    - **Architecture:**
        - Conv1: 3→128 channels, 3×3 kernel, stride 1, padding 1
        - ReLU activation
        - MaxPool: 2×2, stride 2 (dimension reduction)
        - Conv2: 128→256 channels, 3×3 kernel, stride 1, padding 1
        - ReLU activation
        - MaxPool: 2×2, stride 2 (dimension reduction)
        - Conv3: 256→512 channels, 3×3 kernel, stride 1, padding 1
        - ReLU activation
        - MaxPool: 2×2, stride 2 (dimension reduction)
        - FC1: 512×20×20 → 1024
        - ReLU activation
        - FC2: 1024 → 10 (output classes)

- **Training Parameters:**
  - Learning rate: 0.001
  - Optimizer: Adam (β_1=0.9, β_2=0.999)
  - Batch size: 128
  - Epochs: 50
  - Loss function: Cross-Entropy

2. **Model with Batch Normalization**
   - **Additional Components:**
     - BatchNorm2d after each convolutional layer
     - BatchNorm2d after first fully connected layer
   - **Implementation Details:**
     - Running statistics tracked during training
     - Applied before ReLU activation
3. **Model with Dropout**
   - **Dropout Placement:**
     - After each MaxPool layer
     - After first fully connected layer
   - **Implementation Strategy:**
     - Monte Carlo Dropout during training
     - Disabled during evaluation
     - Scaling of activations during training
   - **Training Adjustments:**
     - Learning rate: 0.001
     - Weight decay: 1e-4 (additional regularization)
4. **Model with Data Augmentation**
   - **Augmentation Techniques:**
     - Random horizontal flip
     - Random color jitter:
       * Brightness: ±0.2
       * Contrast: ±0.2
       * Saturation: ±0.2
   - **Implementation Details:**
     - Applied only during training
     - Transforms composed in random order
     - Normalization applied after augmentation
   - **Training Parameters:**
     - Increased batch size to 128
     - Learning rate: 0.0001
     - Epochs: 50 (more training time needed)
5. **Combined Model (All techniques)**
   - **Architecture Integration:**
     - Batch normalization after each conv layer
     - Dropout after pooling (p = 0.25) and FC1 (p = 0.5)
     - Full data augmentation pipeline
   - **Training Strategy:**

  - Batch size: 128
  - Epochs: 50
  - Weight decay: 1e-4
- **Performance Monitoring:**
  - Early stopping patience: 10 epochs
  - Model checkpointing
  - Validation accuracy threshold: 0.60

3.4 Comparative Analysis

The implementation of these configurations revealed several key insights:

- Batch normalization significantly reduced training time and improved stability
- Dropout effectively prevented overfitting, particularly in the fully connected layers
- Data augmentation showed the most impact on generalization to unseen data
- The combined approach achieved the best balance between model performance and generalization

Table 1 presents a quantitative comparison of the different configurations:

Configuration	Test Accuracy	Training Time	Convergence Epochs
Baseline	62.03%	14 min	50
With BatchNorm	66.31%	16 min	50
With Dropout	63.52%	15 min	50
With Augmentation	65.01%	17 min	50
Combined	69.80%	20 min	50

Table 1: Performance Comparison of Different Model Configurations

3.5 Results and Analysis

3.5.1 Gradients Analysis for ReLU Activation Function

The initialization of weight parameters is important, alongside the choice of activation function. By default, PyTorch uses Kaiming initialization for linear layers, optimized for ReLU activations. ReLU helps maintain training stability by enabling consistent gradient flow throughout the network. However, because ReLU outputs zero for any negative input, the network's output distribution after linear layers doesn't follow a Gaussian shape but rather has a longer tail toward positive values.

A known limitation of the ReLU activation function is the occurrence of "dead neurons," which are neurons with zero gradients for all training inputs. This issue arises when no gradient is provided to the layer, preventing the network from learning meaningful weights for the previous layer's neuron output. Dead neurons occur when a particular neuron's output in the linear layer before the ReLU activation is negative for all inputs, resulting in a constant zero output for that neuron.

So, to summarize:

1. **Gradient Stability:**
   - ReLU maintains stability during the training process
   - Facilitates uniform gradient propagation through network layers
   - Exhibits a pronounced peak at zero, consistent with theoretical predictions
2. **Distribution Characteristics:**

- Post-linear layer distributions deviate from Gaussian patterns
- Demonstrates asymmetric behavior with extended positive tails
- Zero gradients for negative inputs create distinctive distribution patterns

Dead neurons occur when:

- A neuron consistently outputs zero for all input patterns
- The pre-activation (linear layer output) remains negative across all inputs
- Gradient flow becomes permanently blocked for affected neurons

The dead neuron phenomenon has several implications:

- **Parameter Stagnation:** Weights in previous layers connected to dead neurons cease to update
- **Capacity Reduction:** Effective network capacity decreases as dead neurons no longer contribute
- **Training Inefficiency:** Resources are wasted on maintaining inactive pathways

To address these challenges, several approaches can be considered:

1. **Initialization Techniques:**
   - Proper scaling of initial weights using Kaiming initialization
   - Careful bias initialization to prevent systematic negative outputs
2. **Architecture Modifications:**
   - Use of Leaky ReLU or other variants
   - Implementation of residual connections
   - Proper learning rate scheduling
3. **Training Monitoring:**
   - Regular checking of activation patterns
   - Gradient flow visualization
   - Early detection of dead neurons

Indeed, the baseline model has a significantly lower performance when talking about loss and accuracy on the validation set. Furthermore, it tends to overfit.

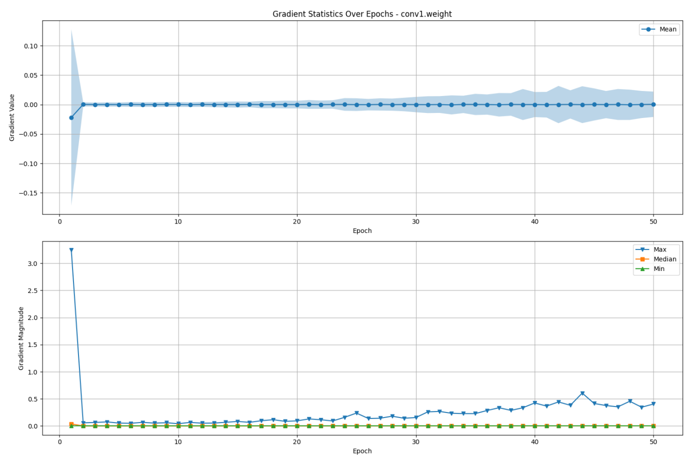


Figure 2: Gradients through the First Convolution Layer

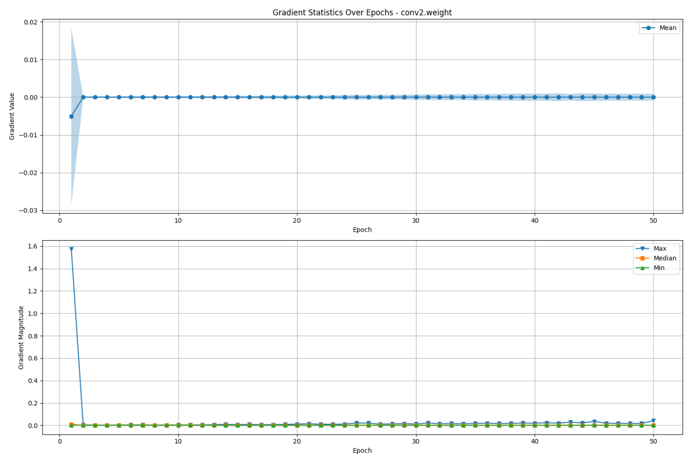


Figure 3: Gradients through the Second Convolution Layer

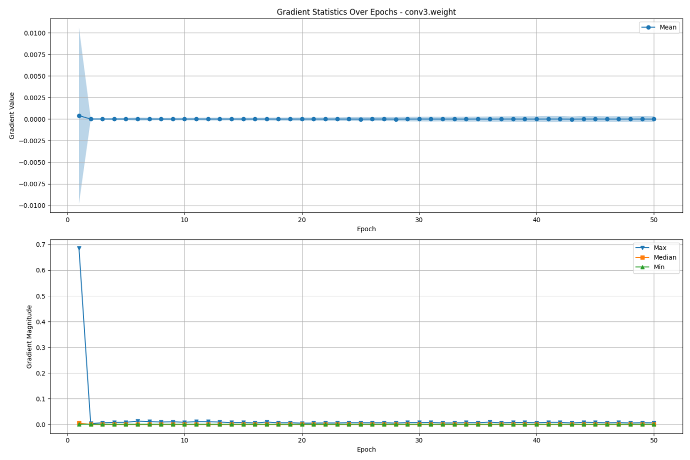


Figure 4: Gradients through the Third Convolution Layer

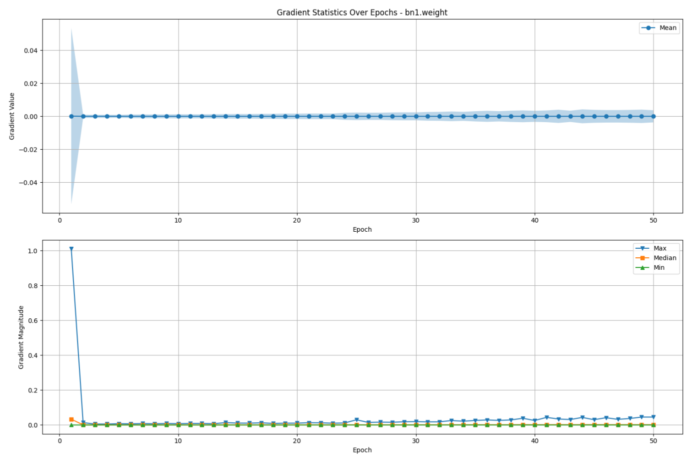


Figure 5: Gradients through the First Batch Normalization Layer

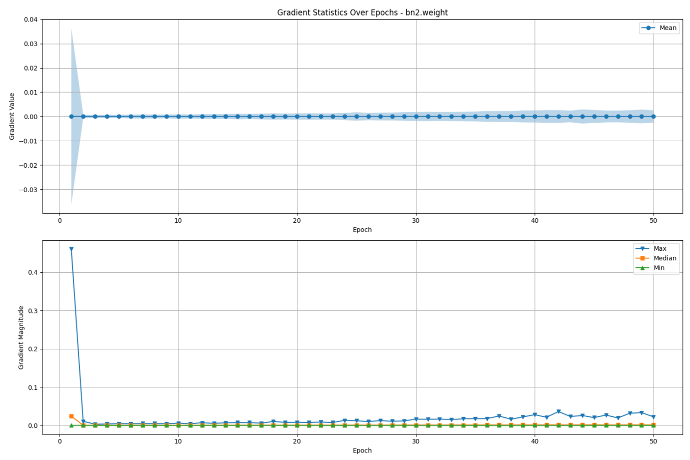


Figure 6: Gradients through the Second Batch Normalization Layer

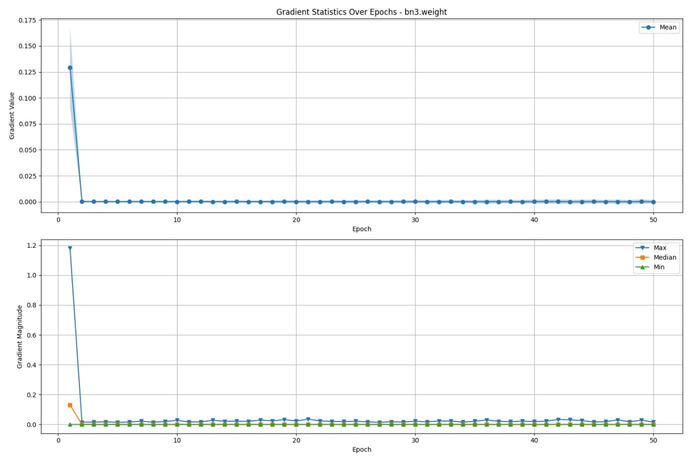


Figure 7: Gradients through the Third Batch Normalization Layer

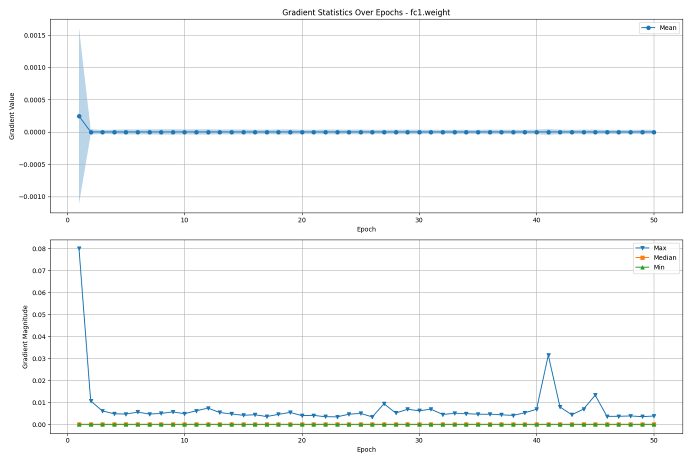


Figure 8: Gradients through the First Fully Connected Layer

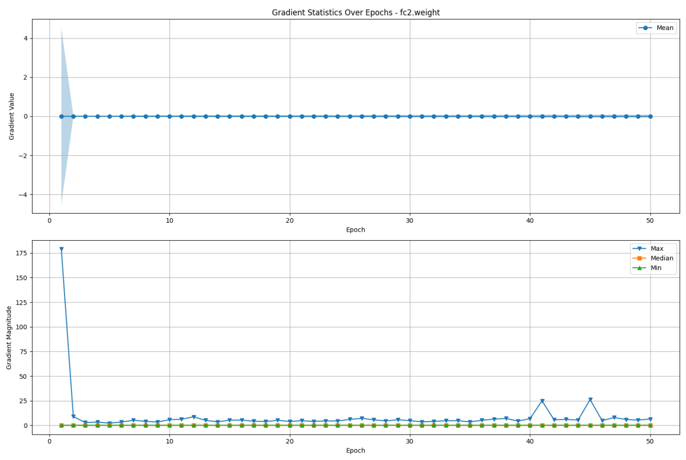
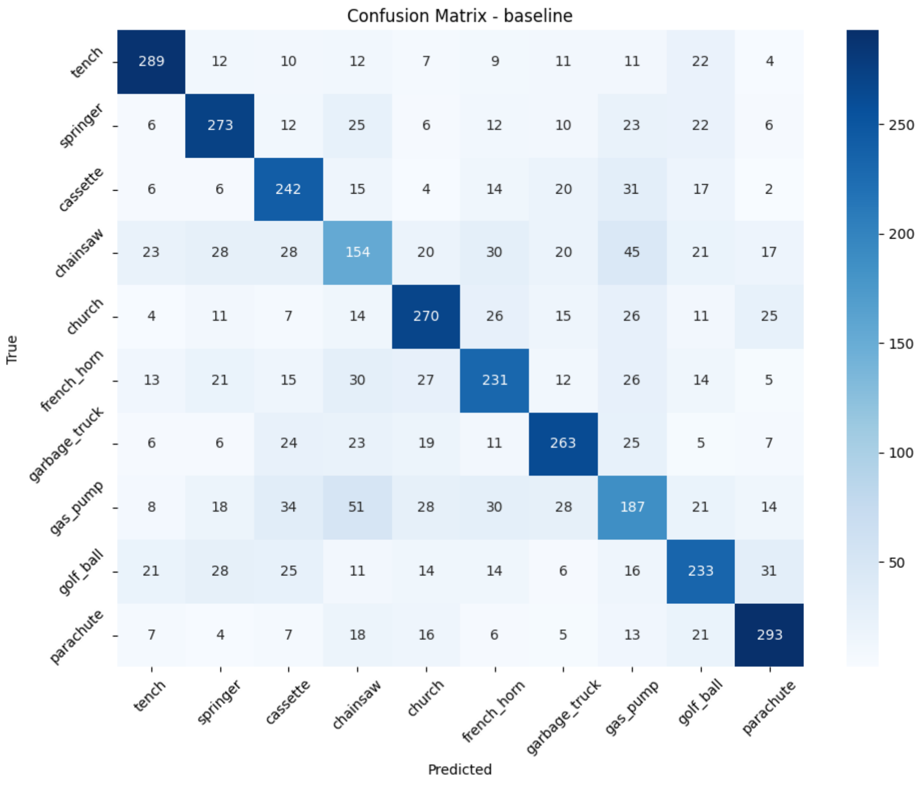


Figure 9: Gradients through the Second Fully Connected Layer

3.5.2 Baseline Performance



	tench	springer	cassette	chainsaw	church	french_horn	garbage_truck	gas_pump	golf_ball	parachute
tench	289	12	10	12	7	9	11	11	22	4
springer	6	273	12	25	6	12	10	23	22	6
cassette	6	6	242	15	4	14	20	31	17	2
chainsaw	23	28	28	154	20	30	20	45	21	17
church	4	11	7	14	270	26	15	26	11	25
french_horn	13	21	15	30	27	231	12	26	14	5
garbage_truck	6	6	24	23	19	11	263	25	5	7
gas_pump	8	18	34	51	28	30	28	187	21	14
golf_ball	21	28	25	11	14	14	6	16	233	31
parachute	7	4	7	18	16	6	5	13	21	293

Figure 10: Confusion Matrix for Baseline Approach

Based on this confusion matrix, it can be observed that the Baseline model struggles to capture the whole semantics of the features (pixels) space (as it misclassifies some of the test images).

3.5.3 Batch Normalization Impact

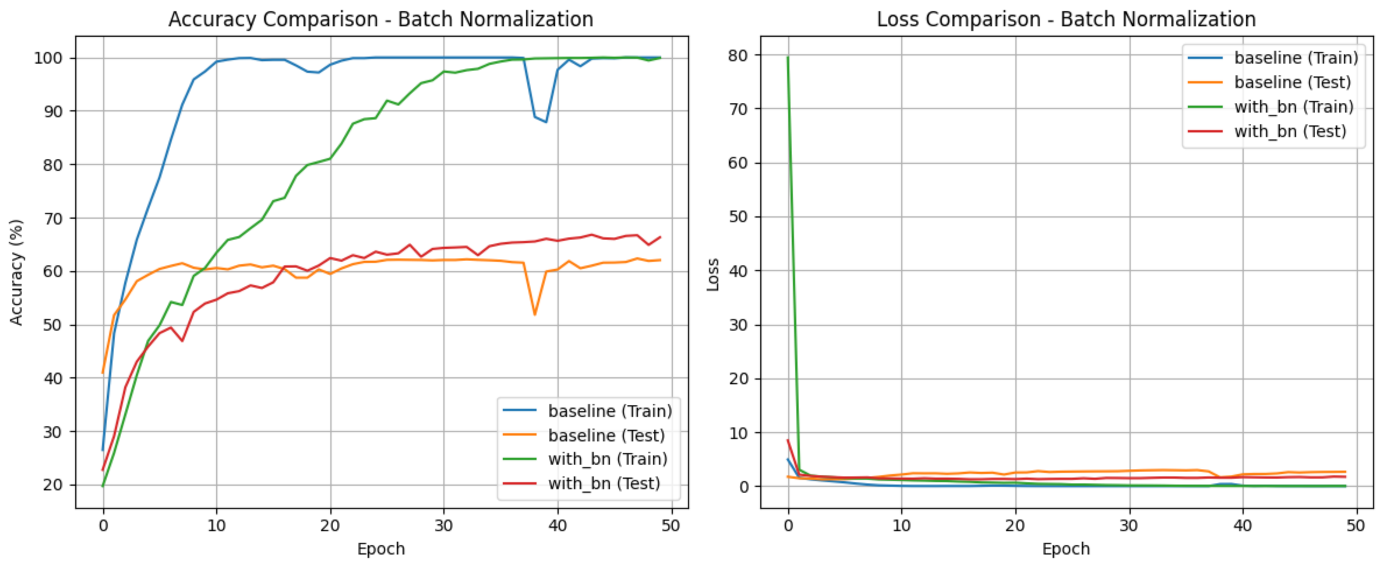


Figure 11: Training and Validation (Accuracy & Loss)

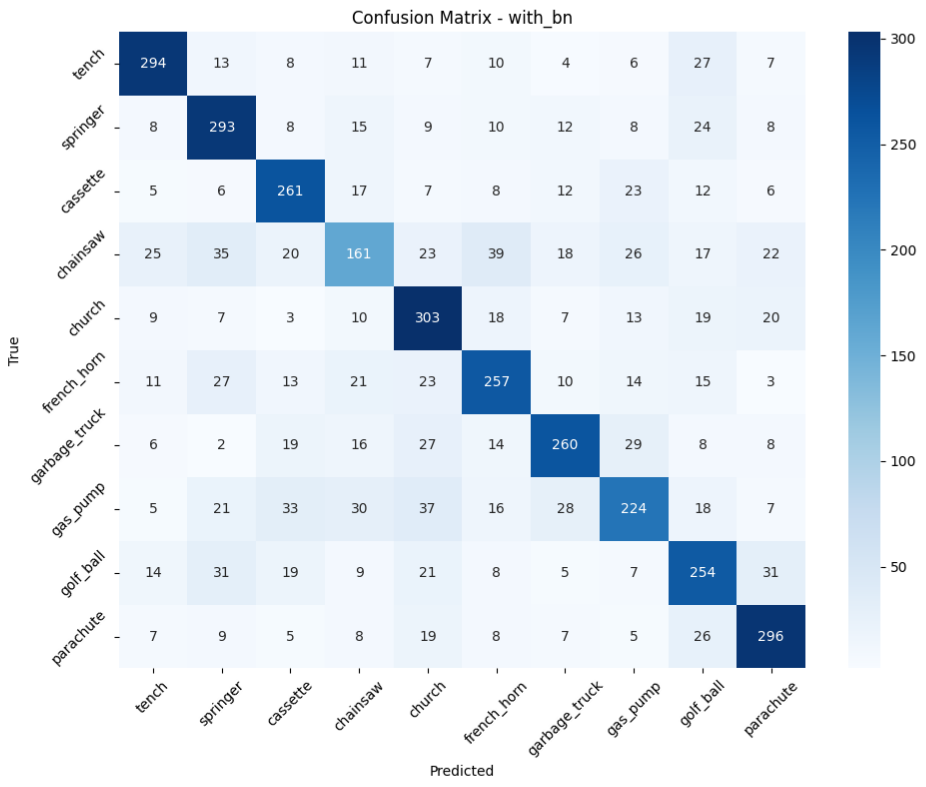


Figure 12: Confusion Matrix for Batch Normalization Approach

3.5.4 Dropout Effects

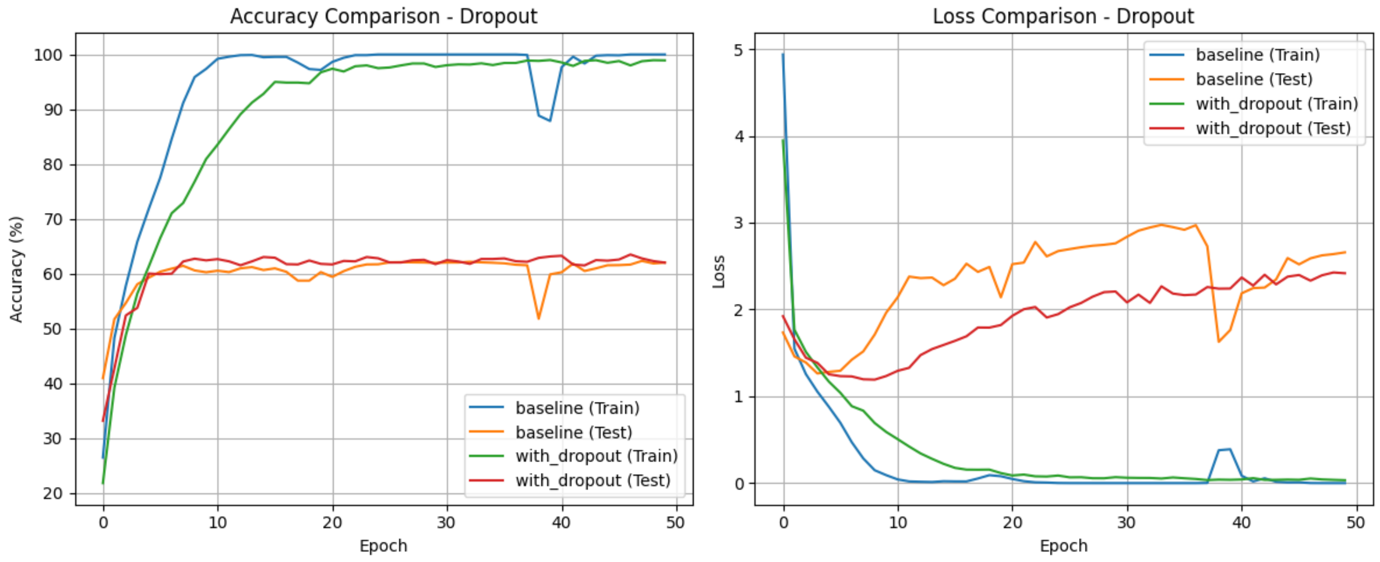
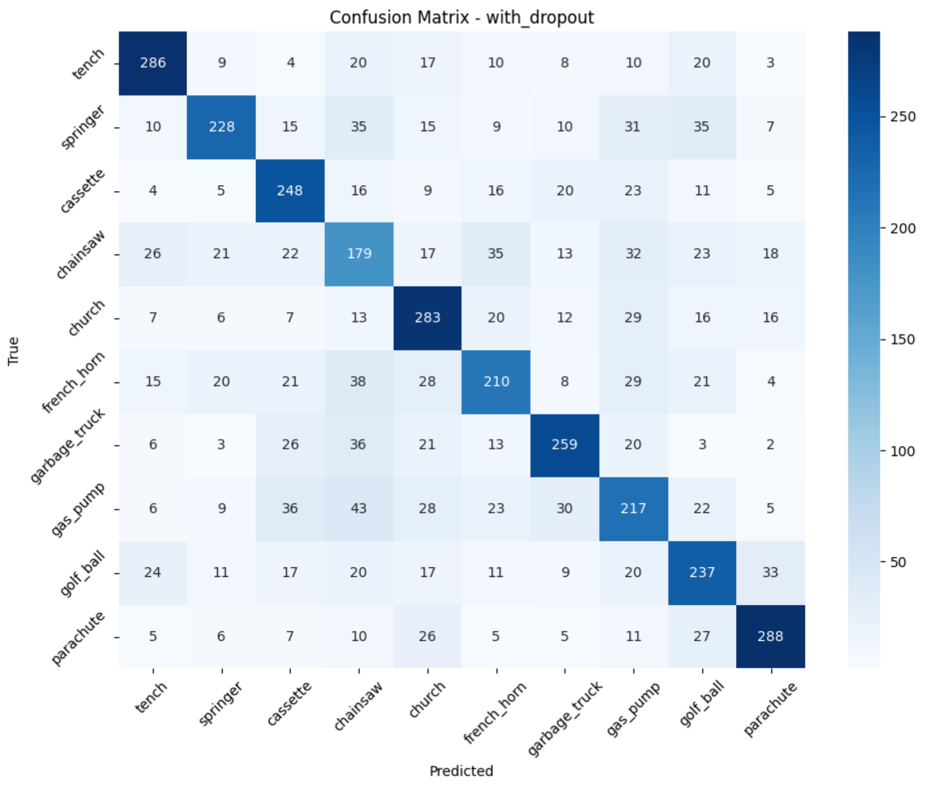


Figure 13: Training and Validation (Accuracy & Loss)



	tench	springer	cassette	chainsaw	church	french_horn	garbage_truck	gas_pump	golf_ball	parachute
tench	286	9	4	20	17	10	8	10	20	3
springer	10	228	15	35	15	9	10	31	35	7
cassette	4	5	248	16	9	16	20	23	11	5
chainsaw	26	21	22	179	17	35	13	32	23	18
church	7	6	7	13	283	20	12	29	16	16
french_horn	15	20	21	38	28	210	8	29	21	4
garbage_truck	6	3	26	36	21	13	259	20	3	2
gas_pump	6	9	36	43	28	23	30	217	22	5
golf_ball	24	11	17	20	17	11	9	20	237	33
parachute	5	6	7	10	26	5	5	11	27	288

Figure 14: Confusion Matrix for Dropout Approach

3.5.5 Data Augmentation Results

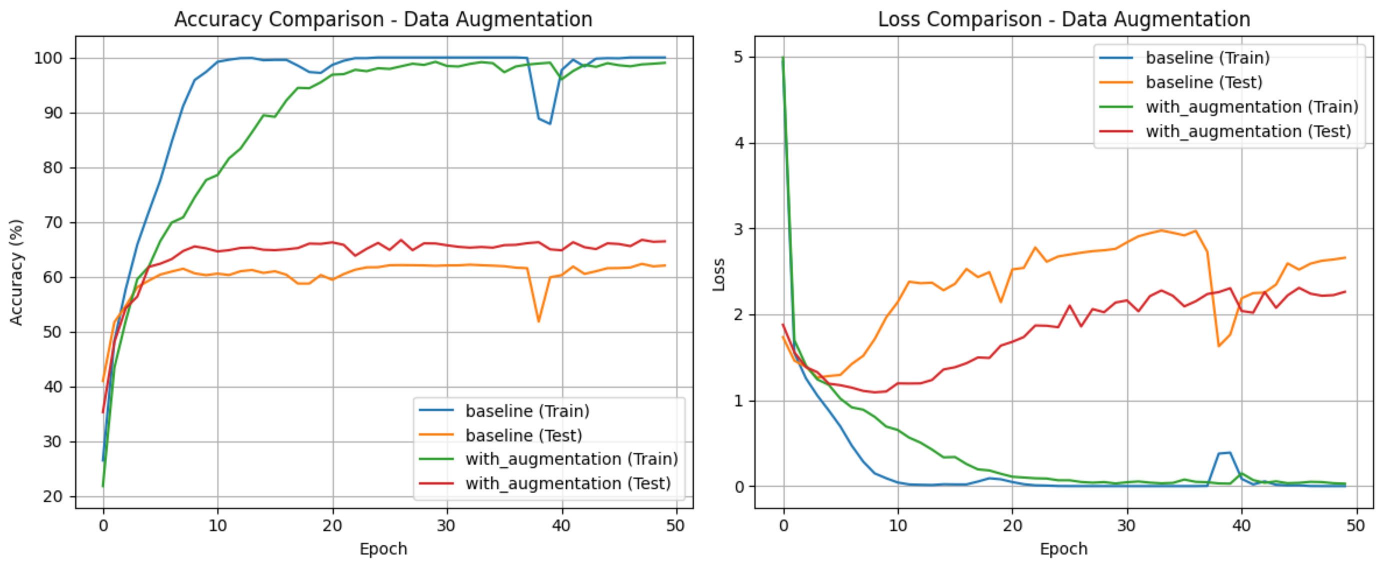
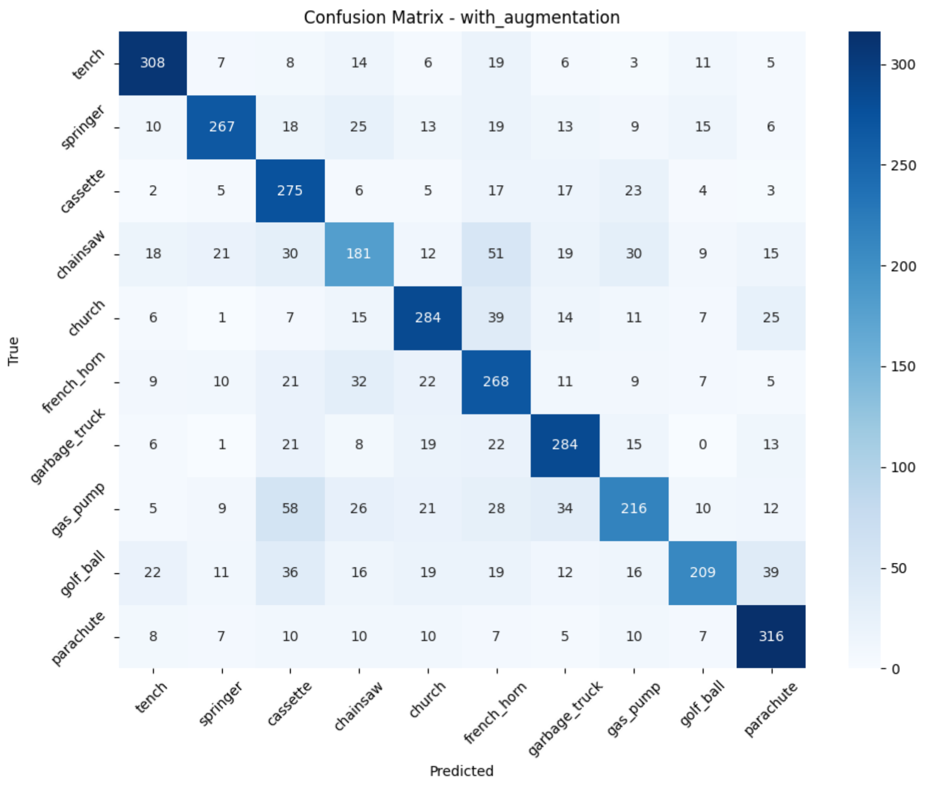


Figure 15: Training and Validation (Accuracy & Loss)



True \ Predicted	tench	springer	cassette	chainsaw	church	french_horn	garbage_truck	gas_pump	golf_ball	parachute
tench	308	7	8	14	6	19	6	3	11	5
springer	10	267	18	25	13	19	13	9	15	6
cassette	2	5	275	6	5	17	17	23	4	3
chainsaw	18	21	30	181	12	51	19	30	9	15
church	6	1	7	15	284	39	14	11	7	25
french_horn	9	10	21	32	22	268	11	9	7	5
garbage_truck	6	1	21	8	19	22	284	15	0	13
gas_pump	5	9	58	26	21	28	34	216	10	12
golf_ball	22	11	36	16	19	19	12	16	209	39
parachute	8	7	10	10	10	7	5	10	7	316

Figure 16: Confusion Matrix for Data Augmentation Approach

3.5.6 Combined Approach

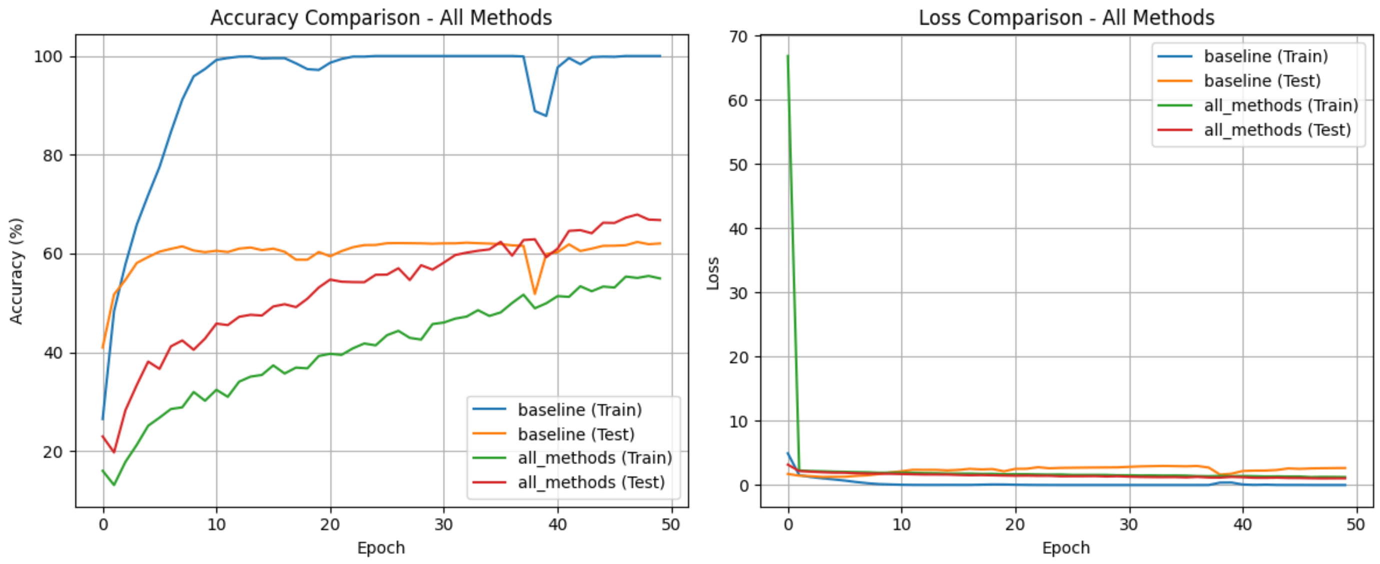
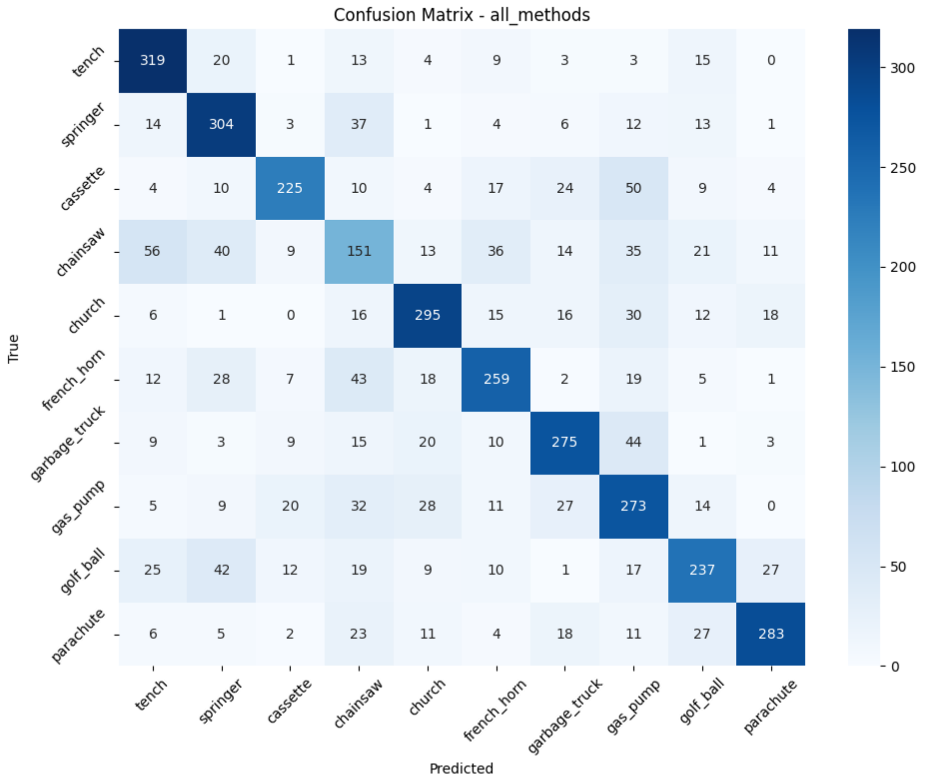


Figure 17: Training and Validation (Accuracy & Loss)



True \ Predicted	tench	springer	cassette	chainsaw	church	french_horn	garbage_truck	gas_pump	golf_ball	parachute
tench	319	20	1	13	4	9	3	3	15	0
springer	14	304	3	37	1	4	6	12	13	1
cassette	4	10	225	10	4	17	24	50	9	4
chainsaw	56	40	9	151	13	36	14	35	21	11
church	6	1	0	16	295	15	16	30	12	18
french_horn	12	28	7	43	18	259	2	19	5	1
garbage_truck	9	3	9	15	20	10	275	44	1	3
gas_pump	5	9	20	32	28	11	27	273	14	0
golf_ball	25	42	12	19	9	10	1	17	237	27
parachute	6	5	2	23	11	4	18	11	27	283

Figure 18: Confusion Matrix for Combined Approach

4 Task 3: Transfer Learning with ResNet-18

4.1 Fine-tuning Strategy

The pre-trained ResNet-18 model was modified by:

- Unfreezing batch normalization layers
- Adapting the final fully connected layer
- Implementing a custom learning rate schedule

4.2 Transfer Learning Results

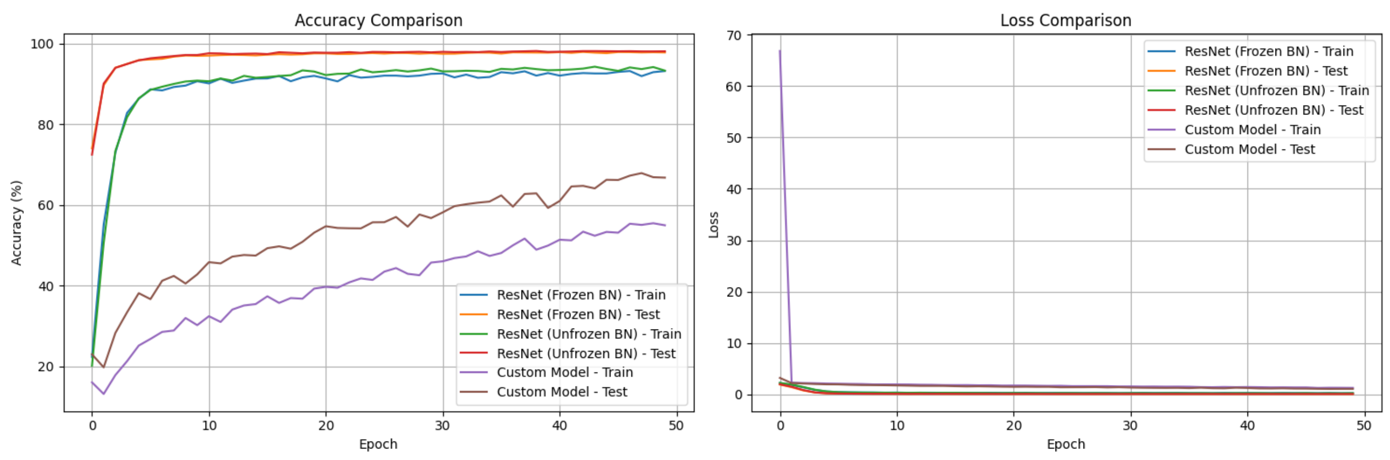


Figure 19: Training and Validation (Accuracy & Loss) for Fine-tuned ResNet-18 and Combined Approach model

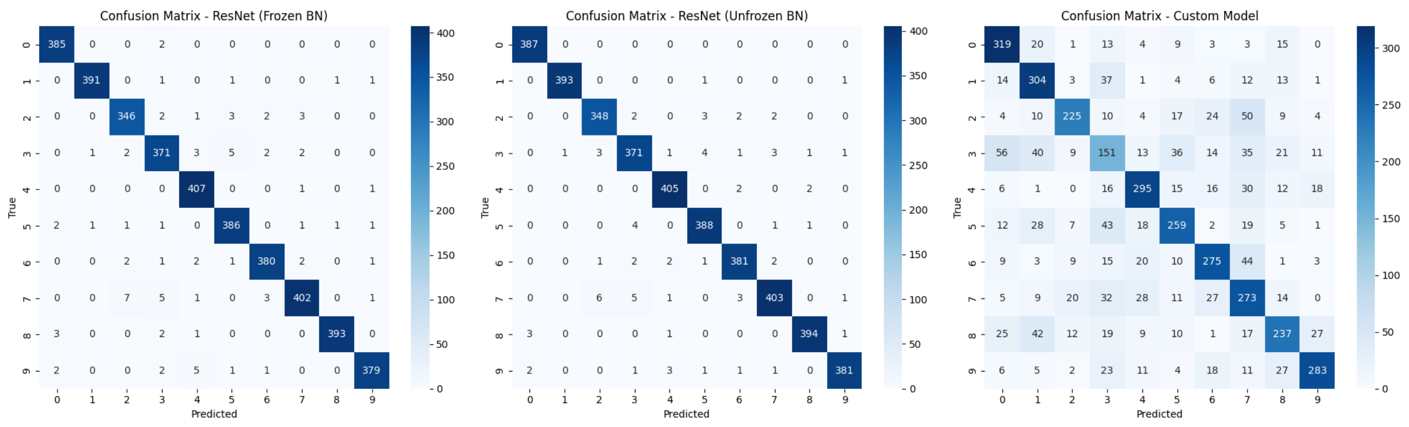


Figure 20: Confusion Matrix for Fine-tuned ResNet-18 and Combined Approach model

As we can see, when talking about freezing and unfreezing the BN layers for ResNet-18, there is no significant difference, in terms of performance. So, to answer the question from this task, the unfreezing of the BN layers helps the model to gain more performance, but this gain is very small.

5 Conclusions

Those experimental results demonstrate the effectiveness of various CNN optimization techniques:

- Batch normalization significantly improved training stability
- Dropout helped reduce overfitting in the fully connected layers
- Data augmentation enhanced model generalization
- Transfer learning with ResNet-18 achieved the highest accuracy

The combined approach with all regularization techniques showed the best performance on the validation set, while transfer learning with the fine-tuned ResNet-18 provided superior results overall.